Introduction to SQL

Class 11

Course Overview

- Introduction to SQL
  - Databases, Tables
  - Classification of SQL – DDL, DML, DCL, TCL
    - DDL – CREATE, ALTER, DROP
    - DML – SELECT, INSERT, UPDATE, DELETE
    - DCL – GRANT, REVOKE
    - TCL – COMMIT, ROLLBACK, SAVEPOINT
  - Data types, Operators
  - Keys – Primary, Foreign, Composite, Unique, Alternate
  - Integrity Constraints – Domain Integrity Constrains, Entity Integrity Constraints, Referential Integrity Constraints
  - Joins – Outer Joins, Left Outer Joins, Right Outer Joins, Inner Joins.
  - Queries, Subqueries, Functions, Flow Control (IF, CASE, WHILE, REPEAT, LOOP), ,Stored functions ,Stored Procedures
  - Views
  - Indexes, Cursors, Triggers, Events
  - Concurrency and locking (Implicit locks, explicit locks, row level locks, table level locks, database level locks)
  - Tuning SQL queries and optimizing performance
  - SQL Databases vs NoSQL Databases
  - ACID, CAP
  - How SQL databases internally works

Stored Functions

- Stored functions in MySQL are used to encapsulate a set of SQL statements into a single, reusable routine (function). They can be used in a variety of situations, including:
  - To perform complex calculations or data manipulations that need to be reused throughout the application.
  - To encapsulate business logic or validation rules that need to be applied consistently across multiple parts of the application.
  - To simplify the application's SQL code by abstracting away complex or repetitive queries.
  - To improve performance by allowing the database to optimize the execution of the stored function.
  - To improve security by allowing the database to execute certain privileged operations without giving the application direct access to the underlying tables.
  - To make the database more modular by breaking it down into smaller, reusable components.

Stored Functions

- Characteristics
  - Stored functions return a single value
  - CREATE ROUTINE database privilege is needed to create a stored function

```
DELIMITER $$

CREATE FUNCTION fun_name(fun_parameter(s))
RETURNS datatype
[NOT] {Characteristics}
fun_body;
```

Returns all the routines from travel database

```
SELECT * FROM information_schema.routines
WHERE routine_schema = 'travel';
```

Stored Functions

- A stored function is a user-defined function that can be used in SQL statements.
- A stored function is defined using the CREATE FUNCTION statement and can take one or more input parameters and return a single value or no value.
- Stored functions can contain SQL statements, control flow statements, and variable assignments.
- Stored functions are similar to stored procedures, but they return a value whereas procedures do not.
- A stored function can be called from a SELECT statement, a SET statement, or from within another stored function or stored procedure.
- Stored functions can be used to encapsulate complex business logic or data validation rules.
- Stored functions are precompiled, which means that the database server performs some optimization on the function when it is created, making its execution faster.
- Stored functions can be used in various contexts, including SELECT, UPDATE, DELETE and INSERT statements.
- Stored functions can be recursive, meaning that a function can call itself, to a certain level.
- Stored functions can be called from triggers, events and views.

Stored Functions

Parameter Name	Descriptions
fun_name	It is the name of the stored function that we want to create in a database. It should not be the same as the built-in function name of MySQL.
fun_parameter	It contains the list of parameters used by the function body. It does not allow to specify IN, OUT, INOUT parameters.
datatype	It is a data type of return value of the function. It should any valid MySQL data type.
characteristics	The CREATE FUNCTION statement only accepted when the characteristics (DETERMINISTIC, NO SQL, or READS SQL DATA) are defined in the declaration.
fun_body	This parameter has a set of SQL statements to perform the operations. It requires at least one RETURN statement. When the return statement is executed, the function will be terminated automatically. The function body is given below: BEGIN -- SQL statements END $$ DELIMITER

Stored Functions - Examples

A function to calculate the total cost of a purchase order:

```
CREATE FUNCTION get_total_cost(order_id INT)
RETURNS DECIMAL(10,2)
BEGIN
  DECLARE total_cost DECIMAL(10,2);
  SELECT SUM(price * quantity) INTO total_cost
  FROM order_items
  WHERE order_id = order_id;
  RETURN total_cost;
END;
```

Stored Functions - Examples

A function to check if an email address is valid:

```
CREATE FUNCTION is_valid_email(email VARCHAR(255))
RETURNS BOOLEAN
BEGIN
  DECLARE email_regex VARCHAR(255) DEFAULT '^[a-zA-Z0-9._%+-
]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$';
  RETURN email REGEXP email_regex;
END;
```

Stored Functions - Examples

A function to calculate the average salary of employees in a department:

```
CREATE FUNCTION get_avg_salary(department_id INT)
RETURNS DECIMAL(10,2)
BEGIN
  DECLARE avg_salary DECIMAL(10,2);
  SELECT AVG(salary) INTO avg_salary
  FROM employees
  WHERE department_id = department_id;
  RETURN avg_salary;
END;
```

Stored Functions - Examples

A function to check if a given date is a weekend or not:

```
CREATE FUNCTION is_weekend(date DATE)
RETURNS BOOLEAN
BEGIN
  RETURN (WEEKDAY(date) = 5 OR WEEKDAY(date) = 6);
END;
```

Stored Functions - Examples

A function to calculate the total number of orders for a customer

```
CREATE FUNCTION get_total_orders(customer_id INT)
RETURNS INT
BEGIN
   RETURN (SELECT COUNT(*) FROM orders WHERE customer_id = customer_id);
END;
```

Stored Functions - Examples

A function to calculate the total number of products in a category:

```
CREATE FUNCTION get_total_products(category_id INT)
RETURNS INT
BEGIN
   RETURN (SELECT COUNT(*) FROM products WHERE category_id = category_id);
END;
```

Stored Functions - Examples

A function to check if a given string is a palindrome or not:

```
CREATE FUNCTION is_palindrome(string VARCHAR(255))
RETURNS BOOLEAN
BEGIN
  RETURN (string = REVERSE(string));
END;
```

Stored Functions - Examples

A function to calculate the total cost of shipping for an order:

```
CREATE FUNCTION get_shipping_cost(order_id INT)
RETURNS DECIMAL(10,2)
BEGIN
   RETURN (SELECT SUM(cost) FROM shipping WHERE order_id = order_id);
END;
```

Stored Functions - Examples

A function to convert lbs to kg

```
DELIMITER //
CREATE FUNCTION lbs_to_kg(lbs MEDIUMINT UNSIGNED)
RETURNS MEDIUMINT UNSIGNED
DETERMINISTIC
BEGIN
  RETURN (lbs * 0.45359237);
END//
DELIMITER ;

SELECT a.plane, max_weight AS max_lbs,
  lbs_to_kg(max_weight) AS max_kg
FROM airplanes a INNER JOIN manufacturers m
  ON a.manufacturer_id = m.manufacturer_id
WHERE m.manufacturer = 'airbus'
ORDER BY a.plane;
```

Stored Functions

- `DROP FUNCTION IF EXISTS lbs_to_kg;`

MySQL IF statements: syntax and examples

- IF-THEN
- IF-THEN-ELSE
- IF-THEN-ELSEIF-ELSE

IF THEN

```
IF condition THEN
  statements;
END IF;
```

MySQL IF-THEN-ELSE statement

```
IF condition THEN
   statements;
ELSE
   else-statements;
END IF;
```

MySQL IF-THEN-ELSEIF-ELSE statement

```
IF condition THEN
   statements;
ELSEIF elseif-condition THEN
   elseif-statements;
...
ELSE
   else-statements;
END IF;
```

MySQL: LOOP Statement

- In MySQL, the LOOP statement is used when you are not sure how many times you want the loop body to execute and you want the loop body to execute at least once.

```
[ label_name: ] LOOP
  {...statements...}
END LOOP [ label_name ];
```

- label_name –
  - Optional. It is a name associated with the LOOP. You use the label_name when executing an ITERATE statement or LEAVE statement.
- You would use a LOOP statement when you are unsure of how many times you want the loop body to execute.
- You can terminate a LOOP statement with either a LEAVE statement or a RETURN statement.

MySQL: LOOP Statement

```
DELIMITER //
CREATE FUNCTION CalcIncome ( starting_value INT )
RETURNS INT
BEGIN
  DECLARE income INT;
  SET income = 0;
  label1: LOOP
    SET income = income + starting_value;
    IF income < 4000 THEN
      ITERATE label1;
    END IF;
    LEAVE label1;
  END LOOP label1;
  RETURN income;
END; //
DELIMITER ;
```

MySQL: REPEAT Statement

- In MySQL, the REPEAT statement is used when you do not know how many times you want the loop body to execute.

[label_name:] REPEAT

{...statements...}

UNTIL condition

END REPEAT [label_name];

- label_name
  - Optional. It is a name associated with the REPEAT loop.
- Statements
  - The statements of code to execute each pass through the REPEAT loop.
- Condition
  - The condition that will terminate the REPEAT loop.
- You would use a REPEAT statement when you are unsure of how many times you want the loop body to execute.
- You terminate a REPEAT statement with the *UNTIL condition*.

MySQL: REPEAT Statement

```
DELIMITER //
CREATE FUNCTION CalcIncome ( starting_value INT )
RETURNS INT
BEGIN
  DECLARE income INT;
  SET income = 0;
  label1: REPEAT
    SET income = income + starting_value;
  UNTIL income >= 4000
  END REPEAT label1;
  RETURN income;
END; //
DELIMITER ;
```

MySQL: WHILE Statement

- In MySQL, the WHILE statement is used when you are not sure how many times you will execute the loop body and the loop body may not execute even once.

[label_name:] WHILE condition DO

  {...statements...}

END WHILE [label_name];

- label_name
  - Optional. It is a name associated with the WHILE loop.
- Condition
  - Te condition is test each pass through the WHILE loop. If the condition evaluates to TRUE, the loop body is executed. If the condition evaluates to FALSE, the WHILE loop is terminated.
- Statements
  - The statements of code to execute each pass through the WHILE loop.
- You would use a WHILE LOOP statement when you are unsure of how many times you want the loop body to execute.
- Since the WHILE condition is evaluated before entering the loop, it is possible that the loop body may not execute even once.

MySQL: WHILE Statement

```
DELIMITER //
CREATE FUNCTION CalcIncome ( starting_value INT )
RETURNS INT
BEGIN
  DECLARE income INT;
  SET income = 0;
  label1: WHILE income <= 3000 DO
    SET income = income + starting_value;
  END WHILE label1;
  RETURN income;
END; //
DELIMITER ;
```

Stored Procedures

- MySQL stored procedures are pre-compiled SQL statements stored in a database
- It is a subroutine or a subprogram in the regular computing language
- **A procedure always contains a name, parameter lists, and SQL statements**.
- We can invoke the procedures by using triggers, other procedures and applications such as Java, Python, PHP, etc.
- There are some tasks which involve running multiple mysql statements. They all can be grouped into one and a stored procedure can be created

Stored Procedure Features

- Stored Procedure increases the performance of the applications. Once stored procedures are created, they are compiled and stored in the database.
- Stored procedure reduces the traffic between application and database server. Because the application has to send only the stored procedure's name and parameters instead of sending multiple SQL statements.
- Stored procedures are reusable and transparent to any applications.
- A procedure is always secure. The database administrator can grant permissions to applications that access stored procedures in the database without giving any permissions on the database tables.

Stored Procedure Syntax

```
DELIMITER &&
CREATE PROCEDURE procedure_name [[IN | OUT | INOUT] parameter_
name datatype [, parameter datatype]) ]
BEGIN
    Declaration_section
    Executable_section
END &&
DELIMITER ;
```

Parameter Name	Descriptions
procedure_name	It represents the name of the stored procedure.
parameter	It represents the number of parameters. It can be one or more than one.
Declaration_section	It represents the declarations of all variables.
Executable_section	It represents the code for the function execution.

MySQL procedure parameter has one of three modes:

IN parameter

It is the default mode. It takes a parameter as input, such as an attribute. When we define it, the calling program has to pass an argument to the stored procedure. This parameter's value is always protected.

OUT parameters

It is used to pass a parameter as output. Its value can be changed inside the stored procedure, and the changed (new) value is passed back to the calling program. It is noted that a procedure cannot access the OUT parameter's initial value when it starts.

INOUT parameters

It is a combination of IN and OUT parameters. It means the calling program can pass the argument, and the procedure can modify the INOUT parameter, and then passes the new value back to the calling program.

How to call a stored procedure?

CALL procedure_name (parameter(s))

Display all students whose marks are greater than 70 and count the total number of students

```
DELIMITER &&
CREATE PROCEDURE get_merit_student ()
BEGIN
   SELECT * FROM student_info WHERE marks > 70;
   SELECT COUNT(stud_code) AS Total_Student FROM student_info;
END &&
DELIMITER ;
```

Procedures with IN Parameter

```
DELIMITER &&
CREATE PROCEDURE get_student (IN var1 INT)
BEGIN
   SELECT * FROM student_info LIMIT var1;
   SELECT COUNT(stud_code) AS Total_Student FROM student_info;
END &&
DELIMITER ;

CALL get_student(4);
```

Procedures with OUT Parameter

```
DELIMITER &&
CREATE PROCEDURE display_max_mark (OUT highestmark INT)
BEGIN
   SELECT MAX(marks) INTO highestmark FROM student_info;
END &&
DELIMITER ;

CALL display_max_mark(@M);
SELECT @M;
```

Procedures with INOUT Parameter

```
DELIMITER &&
CREATE PROCEDURE display_marks (INOUT var1 INT)
BEGIN
   SELECT marks INTO var1 FROM student_info WHERE stud_id = var1;
END &&
DELIMITER ;

SET @M = '3';
CALL display_marks(@M);
SELECT @M;
```

How to show or list stored procedures in MySQL?

```
SHOW PROCEDURE STATUS [LIKE 'pattern' | WHERE search_condition]
```

```
SHOW PROCEDURE STATUS WHERE db = 'mystudentdb';
```

How to delete/drop stored procedures in MySQL?

DROP PROCEDURE [IF EXISTS] procedure_name;

DROP PROCEDURE display_marks;

How to alter the procedure in MySQL?

- MySQL does not allow any command to alter the procedure in MySQL
- It provides a command that is used to change the characteristics of a stored procedure.
- This command may alter more than one change in the procedure but does not modify the stored procedure's parameters or body.
- If we want to make such changes, we must drop and re-create the procedure using the DROP PROCEDURE and CREATE PROCEDURE statement.

How to alter the procedure in MySQL?

```
ALTER PROCEDURE procedure_name [characteristics ...]

characteristics: {
    COMMENT 'string'
  | LANGUAGE SQL
  | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }
}

ALTER PROCEDURE get_merit_student
COMMENT 'It displays all records';
```

Drawbacks of Using Stored Procedures

1. They can be difficult to debug and maintain, especially for complex procedures or procedures that are frequently updated.
2. They are typically specific to a particular database management system, which can make them difficult to port to other systems.
3. They can be less flexible than using dynamic SQL, since they are pre-compiled and cannot easily be modified at runtime.
4. They can be vulnerable to SQL injection attacks if not properly secured.
5. They can decrease the performance of the database in some cases, due to increased network traffic and parsing overhead.
6. Can't be called from non-DBMS clients like Python or Java, need a special driver to interact with them.
7. They can be more difficult to test than other types of code, since they are tightly coupled with the database.

Stored Procedures v/s Stored Functions

Functions	Procedures
A function has a return type and returns a value.	A procedure does not have a return type. But it returns values using the OUT parameters.
You cannot use a function with Data Manipulation queries. Only Select queries are allowed in functions.	You can use DML queries such as insert, update, select etc… with procedures.
A function does not allow output parameters	A procedure allows both input and output parameters.
You cannot manage transactions inside a function.	You can manage transactions inside a procedure.
You cannot call stored procedures from a function	You can call a function from a stored procedure.
You can call a function using a select statement.	You cannot call a procedure using select statements.

STORED PROCEDURES example

A procedure to insert a new customer into the database:

```
CREATE PROCEDURE insert_customer(IN first_name VARCHAR(255), IN last_name VARCHAR(255), IN email VARCHAR(255))
BEGIN
  INSERT INTO customers (first_name, last_name, email) VALUES (first_name, last_name, email);
END;
```

STORED PROCEDURES example

A procedure to update the stock level of a product:

```
CREATE PROCEDURE update_stock(IN product_id INT, IN stock_change INT)
BEGIN
  UPDATE products SET stock = stock + stock_change WHERE id = product_id;
END;
```

STORED PROCEDURES example

A procedure to delete all orders for a customer:

```
CREATE PROCEDURE delete_customer_orders(IN customer_id INT)
BEGIN
  DELETE FROM orders WHERE customer_id = customer_id;
END;
```

STORED PROCEDURES example

A procedure to update the shipping address for an order:

```
CREATE PROCEDURE update_shipping_address(IN order_id INT, IN address VARCHAR(255))
BEGIN
  UPDATE orders SET shipping_address = address WHERE id = order_id;
END;
```

STORED PROCEDURES example

A procedure to calculate the total sales for a given date range:

```
CREATE PROCEDURE get_total_sales(IN start_date DATE, IN end_date DATE)
BEGIN
  SELECT SUM(total_cost) FROM orders WHERE order_date BETWEEN start_date AND end_date;
END;
```

STORED PROCEDURES example

A procedure to update the price of all products in a category:

```
CREATE PROCEDURE update_category_prices(IN category_id INT, IN price_change DECIMAL(10,2))
BEGIN
  UPDATE products SET price = price + price_change WHERE category_id = category_id;
END;
```

STORED PROCEDURES example

A procedure to retrieve the top 10 best-selling products:

```
CREATE PROCEDURE get_top_sellers()
BEGIN
  SELECT product_name, SUM(quantity) as total_sales
  FROM order_items
  JOIN products ON products.id = order_items.product_id
  GROUP BY product_id
  ORDER BY total_sales DESC
  LIMIT 10;
END;
```

STORED PROCEDURES example

A procedure to transfer all orders from one customer to another:

```
CREATE PROCEDURE transfer_orders(IN old_customer_id INT, IN new_customer_id INT)
BEGIN
  UPDATE orders SET customer_id = new_customer_id WHERE customer_id = old_customer_id;
END;
```

STORED PROCEDURES example

A procedure to retrieve the total number of customers who have placed an order:

```
CREATE PROCEDURE get_total_customers()
BEGIN
  SELECT COUNT(DISTINCT customer_id) FROM orders;
END;
```

STORED PROCEDURES example

A procedure to update the price of a product based on its current stock level:

```
CREATE PROCEDURE update_price_based_on_stock(IN product_id INT)
BEGIN
  DECLARE stock INT;
  SELECT stock INTO stock FROM products WHERE id = product_id;
  IF stock <= 10 THEN
    UPDATE products SET price = price * 1.1 WHERE id = product_id;
  ELSEIF stock > 10 AND stock <= 20 THEN
    UPDATE products SET price = price * 1.05 WHERE id = product_id;
  ELSE
    UPDATE products SET price = price * 0.95 WHERE id = product_id;
  END IF;
END;
```

STORED PROCEDURES example

A procedure to update the shipping cost of an order based on the total cost:

```
CREATE PROCEDURE update_shipping_cost(IN order_id INT)
BEGIN
  DECLARE total_cost DECIMAL(10, 2);
  SELECT total_cost INTO total_cost FROM orders WHERE id = order_id;
  UPDATE orders SET shipping_cost =
    CASE
      WHEN total_cost < 50 THEN 5.99
      WHEN total_cost >= 50 AND total_cost < 100 THEN 9.99
      ELSE 14.99
    END
  WHERE id = order_id;
END;
```

STORED PROCEDURES example

A procedure to update the category of a product based on its price:

```
CREATE PROCEDURE update_category_based_on_price(IN product_id INT)
BEGIN
  DECLARE price DECIMAL(10, 2);
  SELECT price INTO price FROM products WHERE id = product_id;
  UPDATE products SET category_id =
    CASE
      WHEN price < 20 THEN 1
      WHEN price >= 20 AND price < 50 THEN 2
      WHEN price >= 50 AND price < 100 THEN 3
      ELSE 4
    END
  WHERE id = product_id;
END;
```

STORED PROCEDURES example

A procedure to update the status of an order based on its shipping date:

```
CREATE PROCEDURE update_order_status(IN order_id INT)
BEGIN
  DECLARE shipped_date DATE;
  SELECT shipped_date INTO shipped_date FROM orders WHERE id = order_id;
  UPDATE orders SET status =
    CASE
      WHEN shipped_date IS NULL THEN 'pending'
      WHEN shipped_date <= CURDATE() THEN 'shipped'
      ELSE 'delivered'
    END
  WHERE id = order_id;
END;
```

STORED PROCEDURES example

A procedure to calculate the discount for a customer based on the total amount of their purchases.

```
CREATE PROCEDURE get_discount(IN customer_id INT)
BEGIN
  DECLARE total_purchase DECIMAL(10, 2);
  SELECT SUM(total_cost) INTO total_purchase FROM orders WHERE customer_id = customer_id;
  SELECT
    CASE
      WHEN total_purchase < 1000 THEN 0
      WHEN total_purchase >= 1000 AND total_purchase < 5000 THEN 5
      ELSE 10
    END as discount;
END;
```

Connect with me on LinkedIn

Please subscribe to our YouTube channel

Check out my GitHub profile

Follow me on Twitter(X)

Thank you